Finite State Models and Natural Language

Roger Levy

Massachusetts Institute of Technology
Department of Brain & Cognitive Sciences

Overview

These slides cover:

- Regular expressions (a minimal summary)
- A bit about phonotactics
- Finite-state automata
- The idea of writing grammar fragments
- Regular languages
- A bit about finite-state transducers
- Weak vs. Strong Generative Capacity
- The limits of finite-state models for natural language syntax

Regular expressions – a minimal characterization

- Given a finite alphabet Σ, any character sequence from $\Sigma \cup \{*, |, (,)\}$ with matching parentheses is a valid regular expression.
- For example, if $\Sigma = \{\texttt{a}, \texttt{b}\}$, the following are valid regular expressions:

`a`	`bab`	`b*a`
`a*`	`ab*`	`(ab)*`
`(a|b)`	`(a*|b)`	`(ab)|(b(a*))`

- The empty string is also a valid regular expression
- Semantics of what each part of a regular expression **matches**:

The empty string	A zero-character sequence
Any character from Σ	That character
Concatenation: XY	what X matches followed by what Y matches
$X*$	0 or more repetitions of X ($*$ is the "Kleene star")
$X \mid Y$	X or Y
()	determine operator precedence

Additional machinery in many regex implementations

- **Partial** (cf. **total**) matching (e.g., Python's `re`'s `search()` vs. `fullmatch()`)
- Conveniences that **don't** change formal power:

`[]`	**Character classes**, e.g. `[a-f]` for `a|b|c|d|e|f`
`^` and `$`	**Anchors** requiring certain in-string position for partial matches
X`+`	"Kleene plus", equiv. $XX*$: 1+ repetitions of X
X{m,n}	**Counters**: between m and n repetitions of X
`.`	**Wildcard** (matches any symbol; a kind of character class)

- **Lookahead** and **lookbehind** guide the regex engine's matcher:

X`(?=`Y`)`	Require that Y follows X in order to match X
X`(?!`Y`)`	Require that Y not follow X in order to match X

- One common extension **does** change formal power: **backreferences**.

`(`X`)`Y`\1`	X then Y then a repetition of the string matching X, e.g.:
`([ab]*)b\1`	Matches `b`, `aba`, `bbb`, `abbab`, ...; doesn't match `abb`, `abbba`, ...

  Matching regexes with arbitrary backreferences is NP-complete (Aho, 1990)!
- Here, we cover regexes *without* backreferences.

To finite-state automata through phonotactics

- **Phonotactics** are the language-specific rules of what sound sequences constitute licit vs. illicit wordforms
- Example:

prick	A word of English
plick	Not a word of English, but *could in principle be one*
pnick	Not a word of English, and "could not be"

Language specificity of phonotactic rules

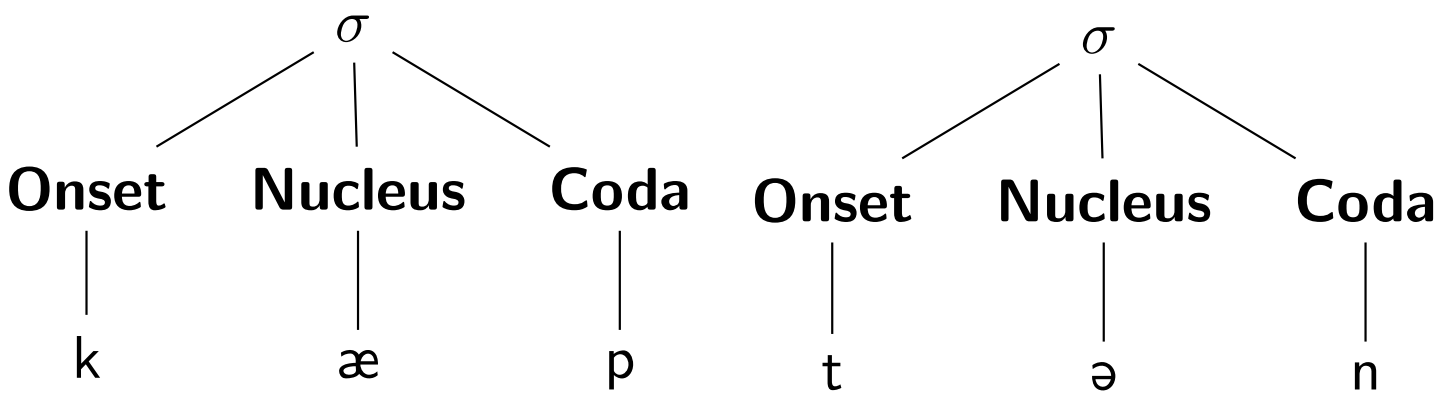
- *zvoon* (in the International Phonetic Alphabet: /zvun/) would be hard-pressed to be a word of English
- But it would be a very natural word in Russian!
- The individual sounds are all in English:

  z as in "zebra"
  v as in "victory"
  u as in "hoop"
  n as in "can"

  but the arrangement into *this sequence* is not OK in English
- Can you think of similar examples involving English and another language you know?

The grammar of English onsets

- Basic syllable structure of English, e.g. for *captain* /kæptən/:



- The badness of *zvoon* has to do with the **onset**
- Some examples of prohibitions on English onsets:
  - **sonorants** (nasals like n, and liquids like l and r) can appear only at the *end* of an onset, e.g.:

    OK: net, ring, bring, plain
    Not OK: nzap, rwell, lroom
  - The only fully acceptable sound that can precede a nasal or **obstruent** (stops like p, fricatives like z, and affricates like ch /tʃ/) is s, e.g.:

    OK: spill, snout
    Not OK: shpill, znout
- Note that these prohibitions vary in generality!

Sound classes in phonology

- Similar to character classes in regexes, e.g., `[A-Fa-f]`, phonologists write sound classes using **phonological features**
- For any feature, a phoneme's value can be $+$, $-$, or unspecified

Feature	+ phonemes	− phonemes
Voice	b, d, ð (first sound of *the*), g, v, z, j, ʒ (second consonant of *measure*)	tʃ ("ch"), f, k, p, s, ʃ ("sh"), t, θ ("th"), h
Labial	b, p, f, v, m, w	none
Sonorant	l, m, n, ŋ ("ng"), r, w, y	all others
Strident	tʃ, j, s, ʃ, z, ʒ	d, ð, t, θ, n, l, r
Continuant	ð, f, s, ʃ, θ, v, z, ʒ, h	b, tʃ, d, g, j, k, p, t

- We can now write phonotactic prohibitions as little regular expressions:

Constraint	Regex	Explanation
$*[\ +\text{Son}\ ][\quad]$	`[mnŋlrwy].`	Sonorants may only be onset-final
$*[\quad][\ +\text{Cont}\ ]$	`.[ðfsʃθvzʒh]`	Fricatives can't have anything before them
$*\begin{bmatrix} +\text{Cont} \\ +\text{Voice} \end{bmatrix}[\quad]$	`[ðvzʒ].`	Voiced fricatives can't precede anything

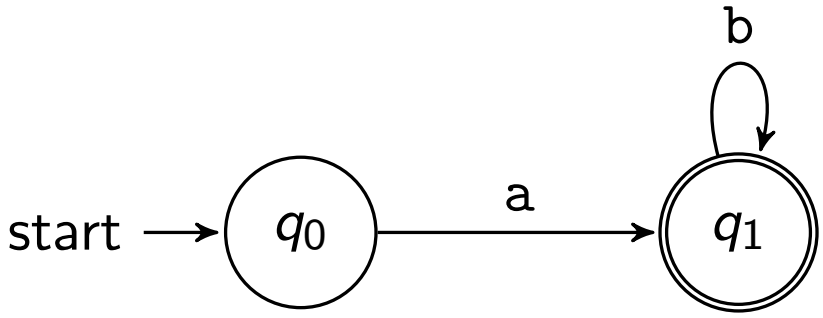
Toward a unified phonotactic grammar

Constraint	Regex	Explanation
$^*[\ +\text{Son}\ ][\quad\quad]$	`[mnŋlrwy].`	Sonorants may only be onset-final
$^*[\quad\quad][\ +\text{Cont}\ ]$	`.[ðfsʃθvzʒh]`	Fricatives can't have anything before them
$^*\begin{bmatrix} +\text{Cont} \\ +\text{Voice} \end{bmatrix}[\quad\quad]$	`[ðvzʒ].`	Voiced fricatives can't precede anything

- We can write each phonotactic **constraint** as a simple regex
- How can we combine a set of phonotactic constraints into a unified **phonotactic grammar**?
- A natural way to do this turns out to be through the formalism of **finite-state machines**

Finite-state automata: an example

- Example **finite-state automaton** (FSA):



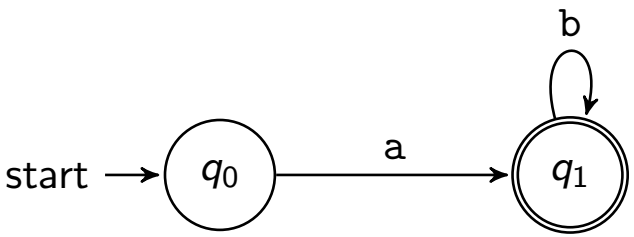
- Accepts all and only those strings that begin with a and then have nothing but b
- More precisely, it accepts all and only the strings accepted by the regular expression ab*

Finite-state automata, formally defined

A **finite-state automaton** consists of:

- A finite set of N **states** $Q = \{q_0, q_1, \ldots, q_{N-1}\}$, with q_0 the **start state**;
- A finite **input alphabet** Σ of symbols (the symbols that comprise strings, like in regexes);
- A set of **final states** $F \subseteq Q$;
- The transition relation Δ, comprised of a finite set of TRANSITIONS each of the form $q \overset{i}{\rightsquigarrow} q'$, with $i \in \Sigma$ or $i = \epsilon$ (ϵ is the **empty string**). Informally, "if you are in state q and have input i available next, you can consume it and move to state q'".

Equivalent specifications of the same FSA



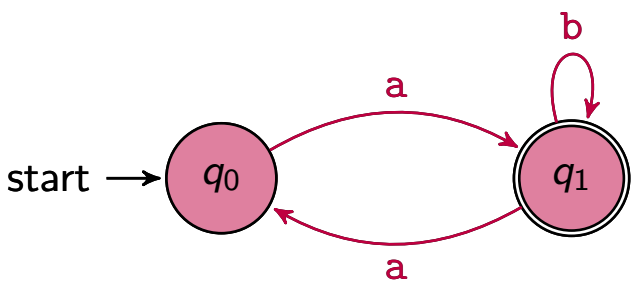
$$Q = \{q_0, q_1\}$$
$$\Sigma = \{\mathtt{a}, \mathtt{b}\}$$
$$F = \{q_1\}$$
$$\Delta = \{q_0 \overset{\mathtt{a}}{\rightsquigarrow} q_1, q_1 \overset{\mathtt{b}}{\rightsquigarrow} q_1\}$$

Acceptance criterion in FSAs (slightly informal)

- An FSA **accepts** a string if you can **recursively** apply the transition relation to the current state (initializing at q_0) and the current position in the string (initializing at the beginning of the string) and get to a final state with the string completely consumed.
- If the sequence of transitions is of length N we may depict a **path through the automaton** that accepts w as

$$q_0 \overset{i_1}{\underset{1}{\rightsquigarrow}} \overset{i_2}{\underset{2}{\rightsquigarrow}} \cdots \overset{i_{N-1}}{\underset{N-1}{\rightsquigarrow}} \overset{i_N}{\underset{N}{\rightsquigarrow}} q^*$$

where $q^* \in F$ and $i_1, \ldots, i_N$ are the appropriately sequenced inputs from w.



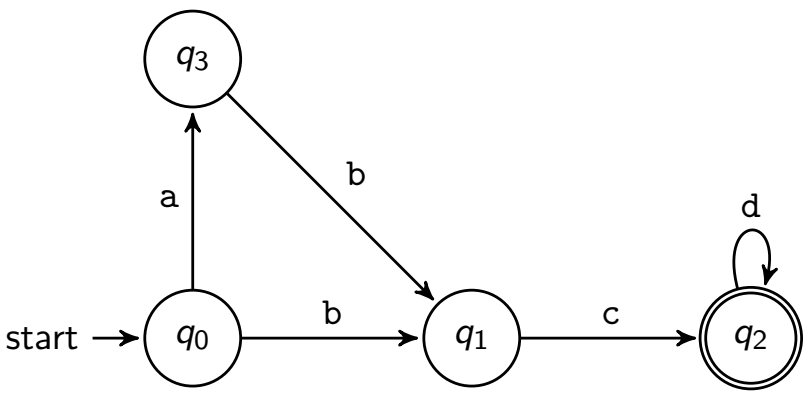
a b b	**Accepted**
b	**Rejected**
a b a	**Rejected**

- **For every regex there is an FSA that accepts all and only the strings the regex matches, and vice versa!**

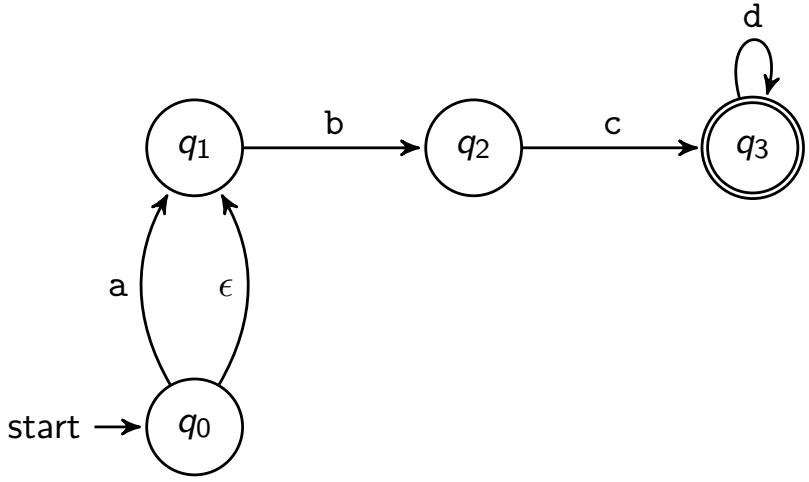
Deterministic versus non-deterministic FSAs

- In a **deterministic** finite-state automaton (DFSA):
  - Each transition's input symbol i must be a symbol in Σ, and cannot be ϵ
  - The transition relation is a **function**.
- As a result, in a deterministic FSA, there is never more than one transition possible given a state and the current position in the string.
- In a **non-deterministic** finite-state automaton (NFSA), neither of those constraints hold.

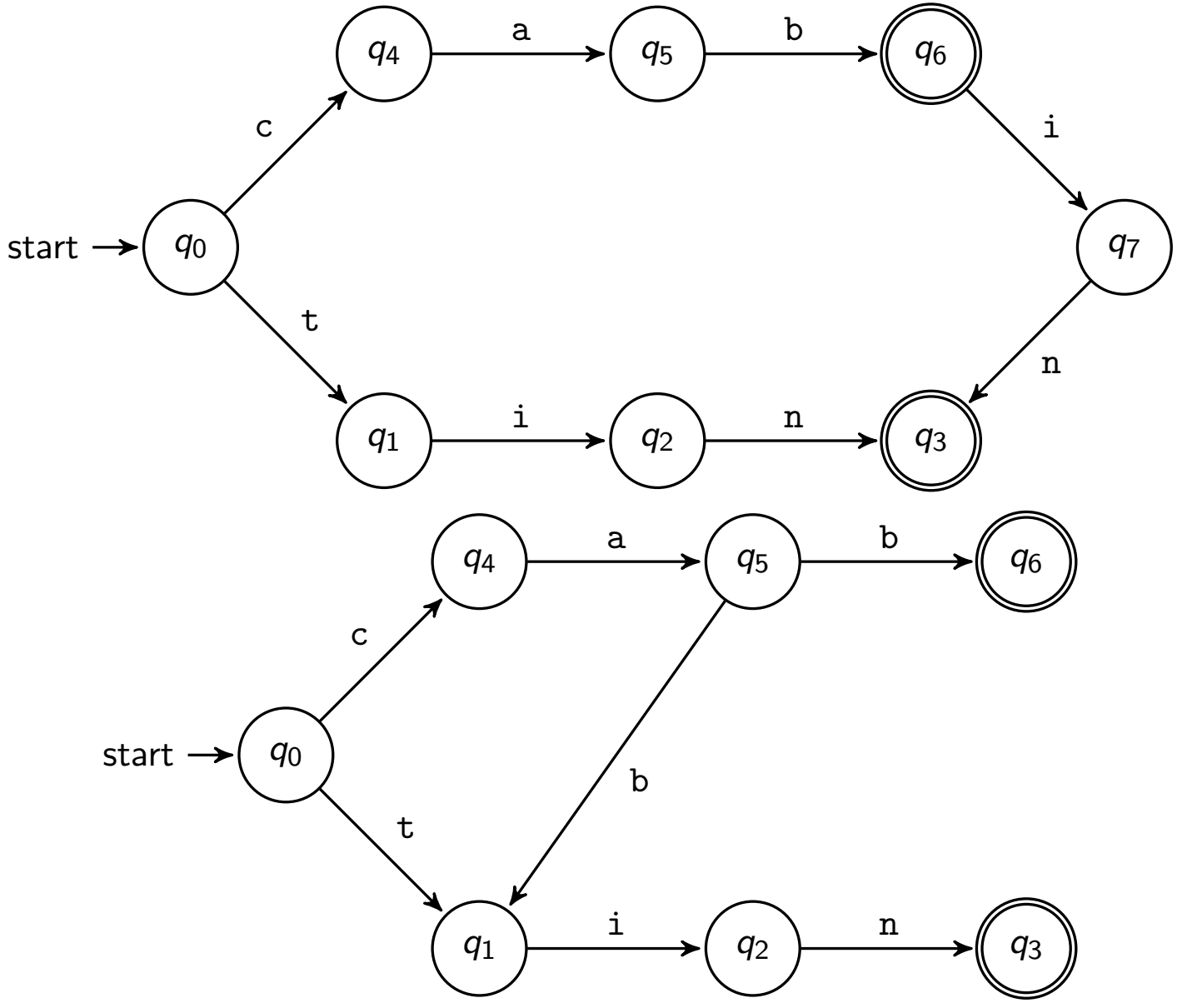
DFSA



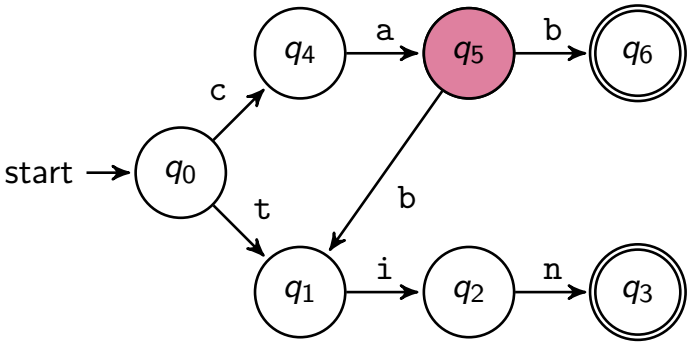
Equivalent NFSA



Another equivalent DFSA/NFSA pair



Checking string acceptance/rejection in an NFSA



c a|b...

- Checking for acceptance is harder for NFSAs than for DFSAs, due to choicepoints!
- Above, there are two possible outward transitions in q_5 for input symbol b.
- Algorithmic options:
  - **Backup:** whenever we encounter a choicepoint, generate a list of transition options and mark our position. Try one. If we fail, go back to the last choicepoint and try the next option on the list. If we run out of options, then the string is rejected.
  - **Lookahead:** look forward in the string to guide choice.
  - **Parallelism:** Instead of maintaining and updating a single state, build a *set* of possible states for each string position.

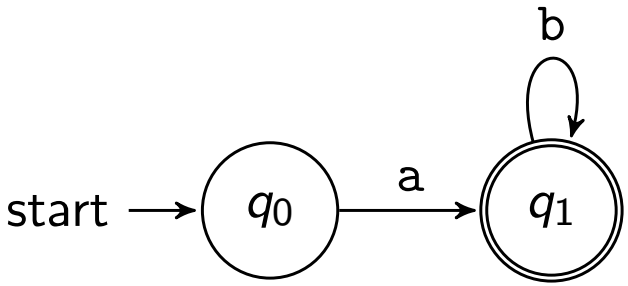
FSA determinization

- Every NFSA can be **determinized** to create a DFSA that accepts and rejects the same strings
- NFSAs and DFSAs are expressively equivalent

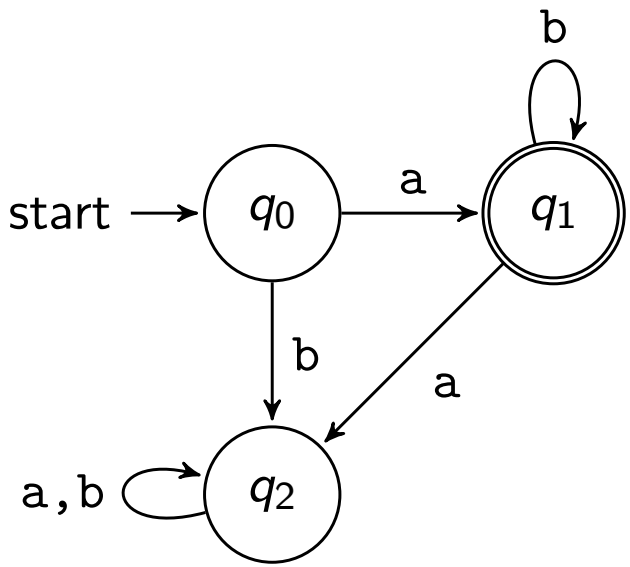
Partial vs total FSAs

- In a TOTAL FSA, state $q \in Q$ and every symbol $i \in \Sigma$, there is a transition $q \overset{i}{\rightsquigarrow} q' \in \Delta$ for some $q' \in Q$.
- Graphically: in every state, for every symbol in Σ there is at least one outgoing arc labeled with that symbol.
- If an FSA is not total, then it is PARTIAL.
- For every partial FSA, there is a total FSA that accepts and rejects exactly the same string set.

Partial FSA



Equivalent total FSA

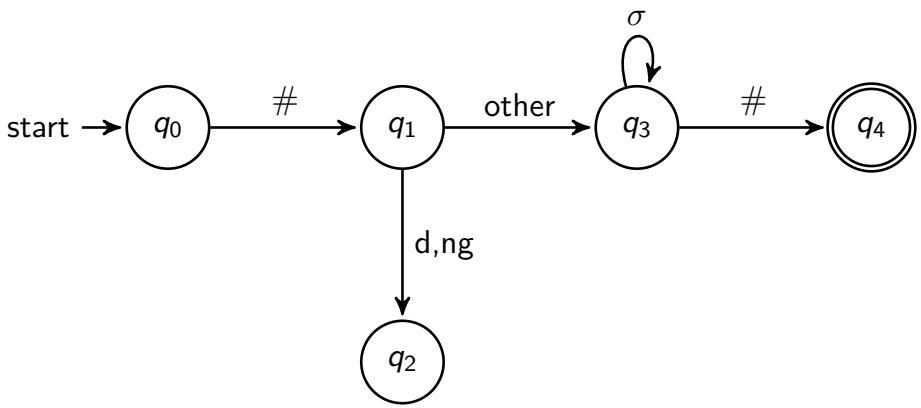


Writing grammar fragments

- With FSAs in hand, we will start to explore writing **fragments** of natural language grammars
- A grammar fragment is not a complete description of a language and its structure
- Rather, a fragment targets *part* of a language and should capture insights about the structure of that part
- Grammar fragments can target any of a number of levels of linguistic structure: lexicon, phonology, morphology, syntax, semantics
- Broad goal: as we accumulate grammar fragments for a language, we should obtain an increasingly close approximation of the *true* characterization of the language and its structure

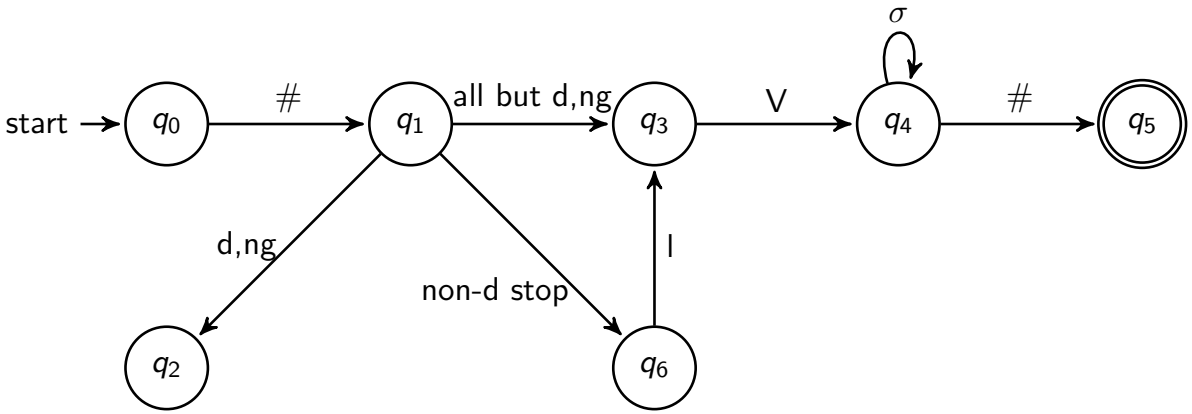
Fragment example 1: Finnish word-level phonotactics

- In Finnish, the possible syllable structures are V, CV, VC, CVC, CCVC (where C=consonant, V=vowel).
- Constraint 1: Word-initially, any consonant can appear except for d and ng.



= word boundary
"other" = symbols in Σ not appearing in another outgoing edge from the state

- Add constraint 2: Word-initially, CC sequences must be stop+liquid(=l).

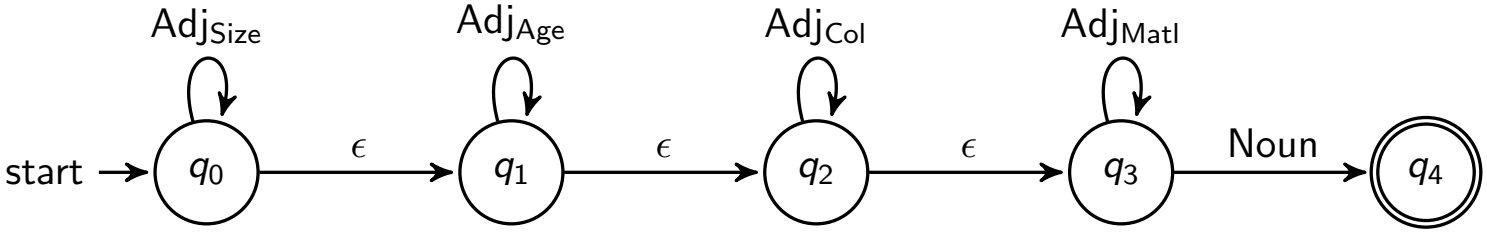


Fragment example 2: English adjective ordering

- Consider the following English adjective classes:

Class	**Examples**
Size	*big*, *short*, *wide*, *heavy*, *voluminous*
Age	*old*, *new*, *recent*
Color	*blue*, *black*, *white*, *colorless*
Material	*wooden*, *organic*, *metal*, *stone*

- The following FSA captures their relative ordering preferences:



Acceptable	**Unacceptable**
table	*old* (missing a noun!)
old table	?*table old*
big blue building	?*blue big building*
voluminous organic produce	?*organic voluminous produce*

- **Note:** the fuller picture is more complicated! For example, contrastive stress can help bring an adjective leftward (e.g., "the **WOODEN** old door, not the **STONE** one"). But this simple description captures some major trends.

Summary thus far

- Regular expressions are an expressive, but constrained, formalism for defining sets of strings
- Finite-state automata are an *expressively equivalent* formalism for defining sets of strings
- We can write **fragments** of natural language grammars using these formalisms for at least some kinds of phonotactics and bits of English syntax
- **Looking ahead:**
  - Mechanisms to combine finite-state grammar fragments into a single unified fragment
  - What parts of natural language structure can and cannot be captured by these formalisms?

Regular languages

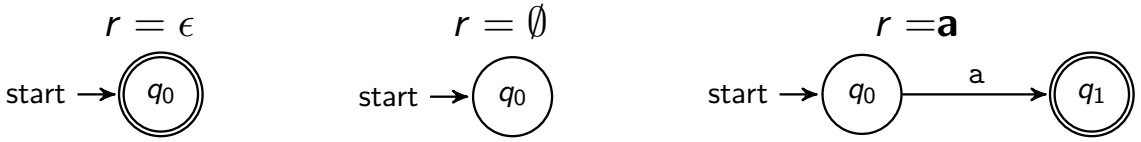
- From the standpoint of formal language theory, a **language** in Σ^* is a set of strings: $L \subseteq \Sigma^*$
- The set of strings matched by a regex, or accepted by an FSA, is the language defined by the regex or FSA
- Any language defined by a regex or FSA is a **regular language**
- The set of **regular languages** for an alphabet Σ can also be defined inductively:
  - $\emptyset$ is a regular language;
  - For all $a \in \Sigma \cup \epsilon$, $\{a\}$ is a regular language;
  - If L is a regular language, then so are:
    - its **complement** $\overline{L} = \{w | w \notin L\}$;
    - Its **Kleene closure**
      $L^* = \{\forall n \in 0, 1, \cdots : w_1 w_2 \ldots w_n | \forall i \in 1 \ldots N : w_i \in L\}$;
  - If L_1 and L_2 are regular languages, then so are:
    - their **concatenation** $L_1 \circ L_2 = \{w_1 w_2 | w_1 \in L_1 \wedge w_2 \in L_2\}$
    - their **union** $L_1 \cup L_2$
- **Recommendation:** compare this inductive definition of regular languages with the syntax & semantics of regular expressions, as they are closely related

Regular languages (standpoint of formal language theory)

- A **language** in Σ^* is a set of strings: $L \subseteq \Sigma^*$
- The set of strings matched by a regex, or accepted by an FSA, is the language defined by the regex or FSA
- Any language defined by a regex or FSA is a **regular language**
- Inductive characterization of the set of **regular languages** for Σ:
  - $\emptyset$ is a regular language;
  - For all $a \in \Sigma \cup \epsilon$, $\{a\}$ is a regular language;
  - If L is a regular language, then so are:
    - its **complement** $\overline{L} = \{w | w \notin L\}$;
    - Its **Kleene closure**
      $L^* = \{\forall n \in 0, 1, \cdots : w_1 w_2 \ldots w_n | \forall i \in 1 \ldots N : w_i \in L\}$;
  - If L_1 and L_2 are regular languages, then so are:
    - their **concatenation** $L_1 \circ L_2 = \{w_1 w_2 | w_1 \in L_1 \wedge w_2 \in L_2\}$
    - their **union** $L_1 \cup L_2$
- **Recommendation:** compare this inductive definition with syntax & semantics of regular expressions, as they're closely related

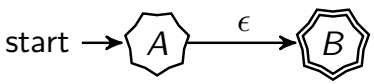
Constructing an FSA from a regex r (slightly informal)

- Base cases:

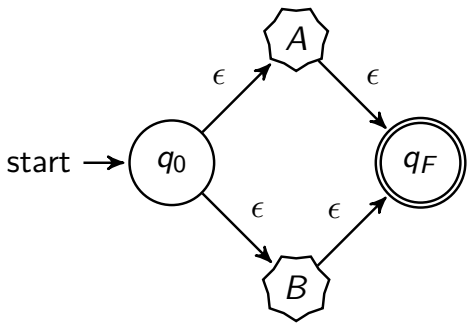


- Additional operators we need for inductive construction:

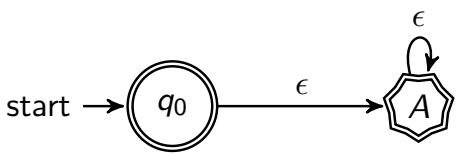
Concatenation



Disjunction



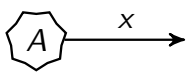
Kleene closure



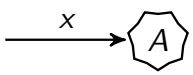
- Semantics regarding the resulting automaton A':

The start state of A is the start state of A'

Every final state of A is final in A'



A' has an x-labeled transition from every final state in A to wherever the arrow points to



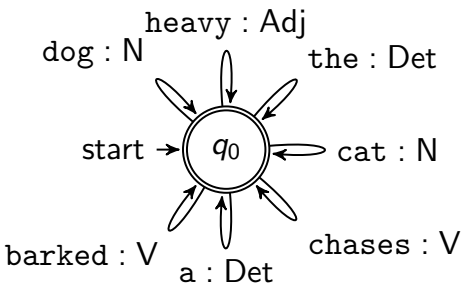
A' has an x-labeled transition from wherever the arrow originates to the start state of A

Closure properties of regular languages

- From the past two slides, we saw that the regular languages are **closed** under:
  - Concatenation
  - Kleene closure
  - Union
  - Complementation
- Notably, this also implies that the regular languages are also closed under **intersection** (Recommended exercise: why?)
- Closure under intersection plays an important role in what's coming up!

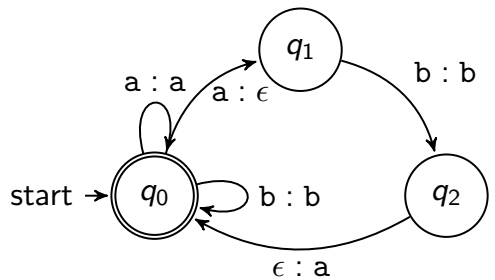
Finite state transducers, briefly

- A **finite-state transducer** consists of:
  - A finite set of N **states** $Q = \{q_0, q_1, \ldots, q_{N-1}\}$, with q_0 the **start state**
  - A finite **input alphabet** Σ of symbols
  - A finite **output alphabet** Γ of symbols
  - A set of **final states** $F \subseteq Q$
  - A **transition relation** Δ: a finite set of **transitions** of the form $q \overset{x:y}{\rightsquigarrow} q'$, with $q, q' \in Q$, $x \in \Sigma \cup \{\epsilon\}$, $y \in \Gamma \cup \{\epsilon\}$. Informally, "if you're in state q and have input x available next, you can output y and move to q'".
- Two example finite-state transducers:



Maps words to parts of speech:

Input	the dog chases the cat
Output	Det N V Det N



Optional exchange of ab

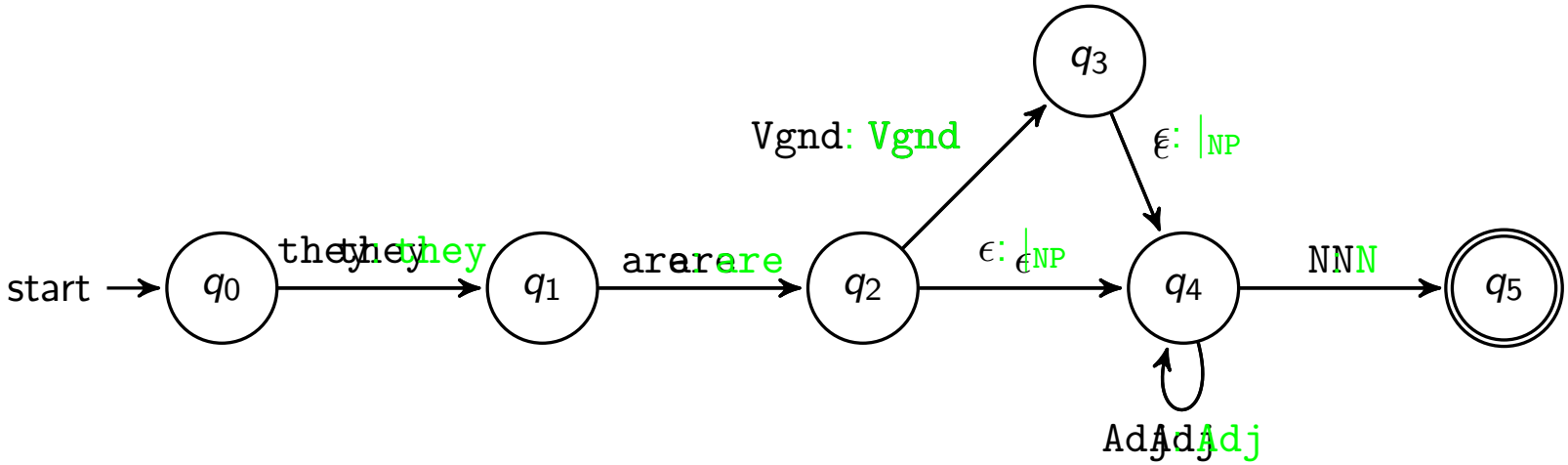
Input	Possible outputs
aabbab	aabbab, ababab
	aabbba, ababba

Structural ambiguity in English syntax

- Consider the following generalizations about English:
  - A sentence can consist of the sequence `They are NP`.
  - A sentence can consist of the sequence `They are Vgnd NP`, where `Vgnd` is a GERUND VERB (*jumping*, *sparkling*, *sleeping*, ...).
  - An **NP** can consist of a noun preceded by zero or more adjectives (e.g., `table`, `precious metals`, `big green buildings`).
  - A gerund verb can function as an adjective inside an NP (e.g., *sleeping children*, *terrific shooting performance*), so that any English word that can serve in the part of speech `Vgnd` can also serve in the part of speech `Adj`.
- **Suggested exercise:** before going on, try to express the following generalizations in an FSA for this fragment of English syntax, over the alphabet $\Sigma = \{\texttt{they}, \texttt{are}, \texttt{Vgnd}, \texttt{Adj}, \texttt{N}\}$

Structural ambiguity in English syntax II

- An FSA that expresses these generalization:



- Implicitly captures a bit of syntax: everything "after" q_4 is part of the post-verbal NP, everything "before" q_4 is outside of it
- We could make this explicit by converting the automaton into a transducer that annotates in the phrase boundary

Input	Output
`they are Vgnd N`	`they are Vgnd` $\vert_{\texttt{NP}}$ `N`
`they are Adj N`	`they are` $\vert_{\texttt{NP}}$ `Adj N`

- The multiple paths through the automaton offer the possibility for different **structural descriptions** of strings

Weak vs. strong generative capacity

- **Weak generative capacity**: what languages (string sets) can be defined by a grammatical formalism?

  `They are Vgnd N`
  `They are Vgnd Adj N`
  `They are Adj N`
  ⋮

- **Strong generative capacity**: what sets of **structural descriptions** can be defined by a grammatical formalism?

  `They are Vgnd` $[_{NP}$ `N` $]_{NP}$
  `They are Vgnd` $[_{NP}$ `Adj N` $]_{NP}$
  `They are Vgnd` $[_{NP}$ `Adj N` $]_{NP}$
  ⋮

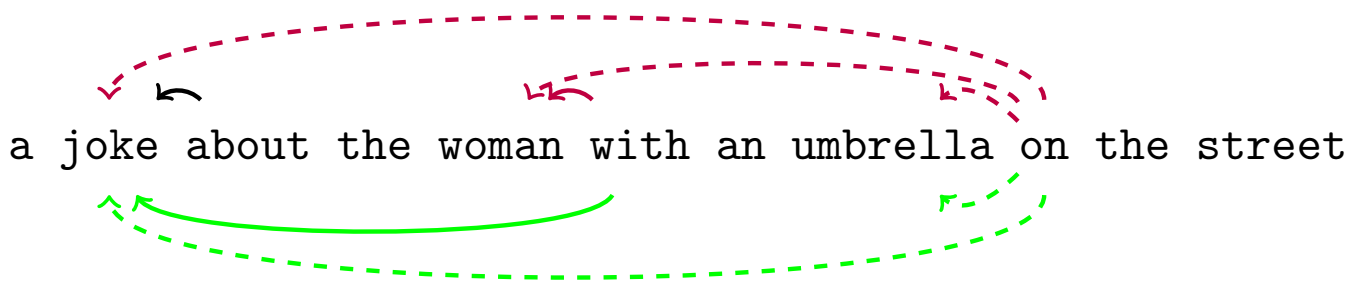
Phenomenon	Finite-state machine (FSM) **weak**?	FSM **strong**?
Gerund/adjective ambiguity	✓	✓

Multiple prepositional phrases in English

- Consider the following set of generalizations:
  - An NP can consist of a determiner and a noun, optionally followed by one or more **prepositional phrase**s (PPs).
  - A PP consists of a preposition followed by an NP.
- Example NPs that these generalizations license:

```
a joke
a joke about the woman
a joke about the woman with an umbrella
a joke about the woman with an umbrella on the street
⋮
```

- **Recommended exercise:** try writing an FSA for this before going on!
- **Observe:** as PPs accumulate, the meanings multiply!



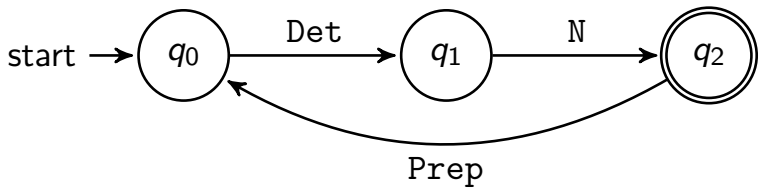
- # meanings grows as the Catalan numbers, $C_k = \binom{2k}{k} - \binom{2k}{k-1}$ (Church & Patil, 1982)

Multiple prepositional phrases in English II

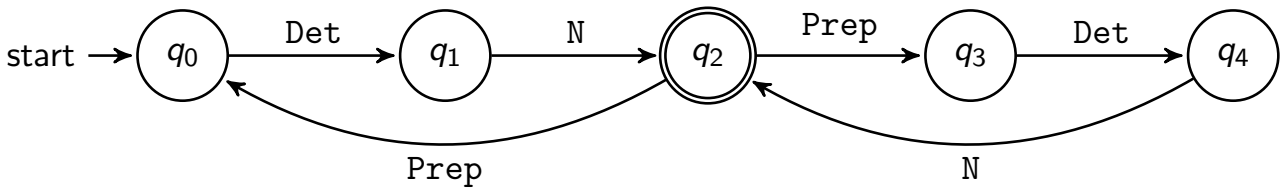
- An NP can consist of a determiner and a noun, optionally followed by one or more PREPOSITIONAL PHRASEs.
- A **prepositional phrase** (PP) consists of a preposition followed by an NP.

```
a joke
a joke about the woman
a joke about the woman with an umbrella
a joke about the woman with an umbrella on the street
```

- **Example FSA one might try:**



- **Problem:** there's only one path, so no mechanism to account for structural ambiguity. We could try adding states. . .



- . . . but we would have to add states for every additional level of PP stacking.
- Since PP stacking is unbounded, a finite-state machine won't be able to generate enough structural descriptions for an unbounded number of PPs.

Weak vs. strong generative capacity

Phenomenon	FSM **weak**?	FSM **strong**?
Gerund/adjective ambiguity	✓	✓
NPs with stacked PP postmodifiers	✓	✗

Case study 3: object-extracted relative clauses

- Consider this sentence of English:

  `the rock that the squirrel likes can be found in the garden`

- Intuitively, it involves combining these two sentences "the right way":
  `the squirrel likes the rock`
  `the rock can be found in the garden`

- It involves the **syntactic construction** of **relativization**, **extracting** the object ~~the rock~~; the resulting **relative clause** is used as a postmodifier:

	rock	that the squirrel likes ~~the rock~~	
the	rock		can be found in the garden
the	rock	that the squirrel likes	can be found in the garden

Multiple center-embedding with relative clauses

	N	that NP V ~~NP~~	
the	N		⟨rest_of_clause⟩
the	N	that NP V	⟨rest_of_clause⟩

- Subject-modifying object-extracted relative clauses can be nested:

  the rock can be found in the garden

  the rock that the squirrel likes can be found in the garden

  the rock that the squirrel that the dog chases likes can be found in the garden

  the rock that the squirrel that the dog that the woman owns chases likes can be found in the garden

  ⋮

- The resulting sentences start to get very hard to understand, but it is theoretically productive to assume that they are implied by the relativization construction and thus part of the language
- That is, they may tax human linguistic **performance**, but they should be part of a theory of human grammatical **competence**

Case study 3: multiple center embedding

```
      N   that NP V NP
the   N                  ⟨rest_of_clause⟩
-------------------------------------------
the   N   that NP V      ⟨rest_of_clause⟩
```

the rock can be found in the garden

the rock that the N V can be found in the garden

the rock that the N that the N V V can be found in the garden

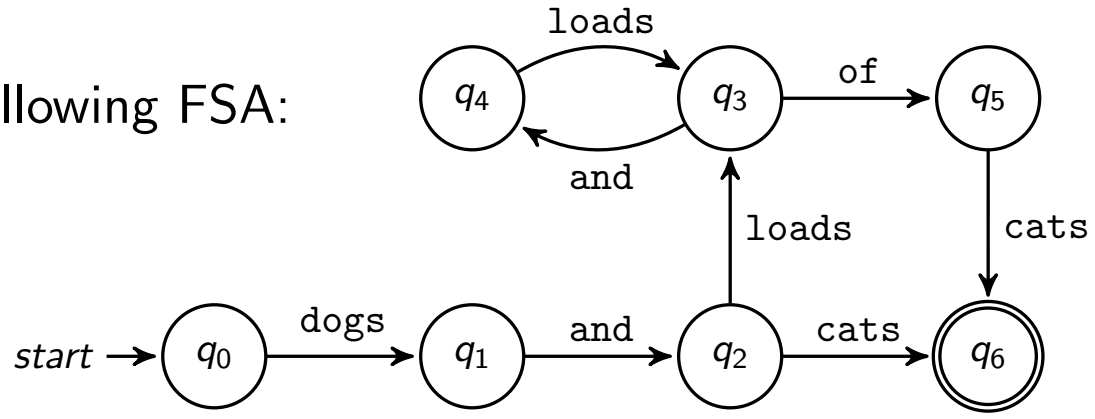
the rock that the N that the N that the N V V V can be found in the garden

⋮

- **Recommended exercise:** before going on, think of how you would try to capture this pattern (at least in part) with a finite-state model.
- It turns out that this pattern *cannot* be modeled with finite-state machines
- Showing this rigorously requires additional technical apparatus that we'll cover next

Pumping strings

- Consider the following FSA:



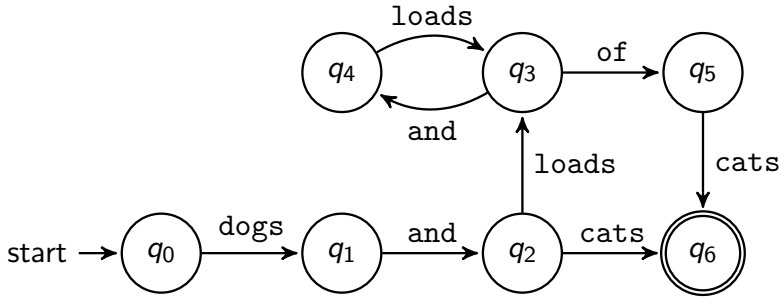
- Corresponding regex: `dogs and (cats|loads (and loads)* of cats)`
- The following sets of strings are accepted:

`dogs and cats` $(=s_1)$
`dogs and loads of cats` $(=s_2)$
`dogs and loads and loads of cats` $(=s_3)$
`dogs and loads and loads and loads of cats` $(=s_4)$
$\vdots$

- For s_3, we could repeat the substring `and loads` as many times as we want!
- This is called **pumping** s_3 with the substring `and loads`.

The Pumping Lemma for regular languages

- Informally: if L is regular, then every string that is "long enough" contains some non-empty "intermediate" section that can be arbitrarily pumped without leaving L.



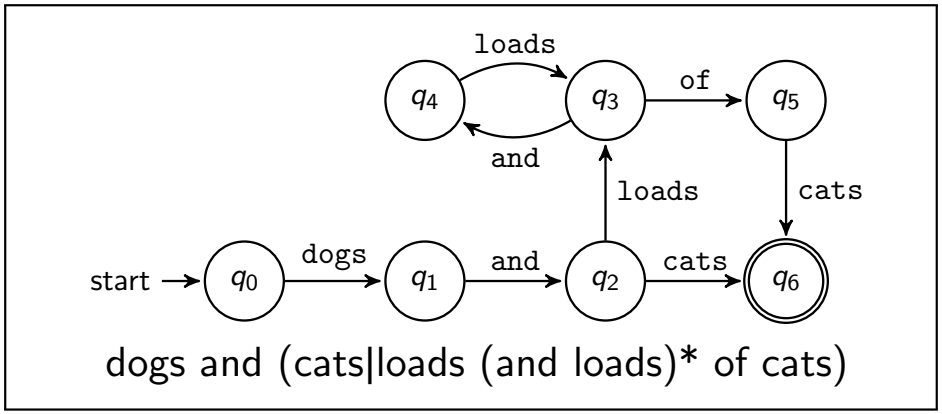
dogs and (cats|loads (and loads)* of cats)

- Formally: if $L \subseteq \Sigma^*$ is regular, then there is some integer k such that for every string $s \in L$ such that $|s| > k$ can be written as $s = xyz$ for $x, y, z \in \Sigma^*$, with:
  - $|y| \geq 1$ (y is non-empty);
  - $|xy| \leq k$;
  - for all $i \geq 0$, $xy^i z \in L$ (y can be pumped in xyz).

Example of the pumping lemma's application

if $L \subseteq \Sigma^*$ is regular, then there is some integer k such that for every string $s \in L$ such that $|s| > k$ can be written as $s = xyz$ for $x, y, z \in \Sigma^*$, with:

- $|y| \geq 1$ (y is non-empty);
- $|xy| \leq k$;
- for all $i \geq 0$, $xy^iz \in L$ (y can be pumped in xyz).



- We can show the Pumping Lemma is satisfied by setting (e.g.) $k = 6$.
- We now need to analyze the infinite set of strings of length > 6, e.g.:

$$\overbrace{\texttt{dogs and loads}}^{x}\ \overbrace{\texttt{and loads}}^{y}\ \overbrace{\texttt{of cats}}^{z}$$

- y is non-empty;
- $|xy| \leq k$;
- xy^iz is in the language for all $i \geq 0$.
- We could do a similar decomposition for every other string in the language of length > 6. Thus, the pumping lemma is satisfied, and the language is regular!

The argument that English is not regular

- We define an idealization of the natural language English and call it $\textsc{English}$.
- We use a regular expression to define a regular language $L^{\ddagger}$ and call its intersection with English $L^{\dagger} = \textsc{English} \cap L^{\ddagger}$.
- If $\textsc{English}$ is regular, then $L^{\dagger}$ must be regular, since regular languages are **closed under intersection**.
- We use the pumping lemma for regular languages to show that $L^{\dagger}$ is *not* regular.
- Therefore, $\textsc{English}$ is not regular.

Evaluating multiple center-embedding with the Pumping Lemma

```
the rock can be found in the garden
the rock that the N V can be found in the garden
the rock that the N that the N V V can be found in the garden
the rock that the N that the N that the N V V V can be found in the
garden
⋮
```

- We will call this infinite set $L^{\dagger}$ and summarize it as:

  $$\texttt{the rock (that the N)}^{i}\ \texttt{V}^{i}\ \texttt{can be found in the garden}$$

  for $i \geq 0$, with the multiple appearances of ^{i} indicating that `(that the N)` and `V` must appear in place the same number of times as each other.
- Note that strings of the following form are not OK English when $i \neq j$:

  $$\texttt{the rock (that the N)}^{i}\ \texttt{V}^{j}\ \texttt{can be found in the garden}$$

  e.g., **the rock that the squirrel can be found in the garden* is not in English.
- We assume that $L^{\dagger}$ is part (formally, a subset) of ENGLISH.

Intersecting ENGLISH with a regular language

$L^{\dagger} =$ `the rock (that the N)`^{i} `V`^{i} `can be found in the garden`

- Call the regular language to the regex below $L^{\ddagger}$:

  `the rock (that the N)* V* can be found in the garden`

- Then

$$\text{ENGLISH} \cap L^{\ddagger} = L^{\dagger}$$

- So, if ENGLISH is regular, then $L^{\dagger}$ is regular!
- We will use the contrapositive: if $L^{\dagger}$ is *not* regular, then ENGLISH is not regular either.
- We will be able to use the Pumping Lemma for regular languages to show that $L^{\dagger}$ is *not* regular

Proving by contradiction that $L^\dagger$ is not regular

if $L \subseteq \Sigma^*$ is regular, then there is some integer k such that for every string $s \in L$ such that $|s| > k$ can be written as $s = xyz$ for $x, y, z \in \Sigma^*$, with:

- $|y| \geq 1$ (y is non-empty);
- $|xy| \leq k$;
- for all $i \geq 0$, $xy^iz \in L$ (y can be pumped in xyz).

- If $L^\dagger$ were regular, then there must be some k per the above.
- Suppose there is some such k. Then consider the following string:

  $$\texttt{the rock (that the N)}^k\ \texttt{V}^k\ \texttt{can be found in the garden}$$

- We should be able to find an appropriate xyz decomposition.
- Since $|xy| \leq k$, y cannot contain any `V`s. We can distinguish two cases:
  - y could contain one or more `N`s. But then pumping y would yield a string that doesn't have the same number of `N`s and `V`s, which wouldn't be in $L^\dagger$!
  - y might be the string `the`, `that`, or `that the`. But then pumping y will also yield a string outside of $L^\dagger$!
- Either way, a contradiction: for ENGLISH to be regular, $L^\dagger$ had to be regular. But $L^\dagger$ can't be regular according to the Pumping Lemma!
- Thus ENGLISH is **not regular**

Summary

Phenomenon	FSM **weak**?	FSM **strong**?
Gerund/adjective ambiguity	✓	✓
NPs with stacked PP postmodifiers	✓	✗
Multiply nested object relative clauses	✗	✗

- Both **weak** and **strong** generative capacity of grammatical formalisms are of interest
- Finite-state models can capture some features of English syntactic structure, but have neither the strong nor the weak generative capacity for other features
- **Looking ahead:** these classic results motivate more expressive grammatical formalisms that have been central to the cognitive science of language for decades

References: general formal language theory I

Harrison, M. A. (1978). *Introduction to formal language theory*. Addison-Wesley Longman Publishing Co., Inc.

Jäger, G., & Rogers, J. (2012). Formal language theory: Refining the chomsky hierarchy. *Philosophical Transactions of the Royal Society B: Biological Sciences*, *367*(1598), 1956–1970.

Kornai, A. (2007). *Mathematical linguistics*. Springer Science & Business Media.

Sipser, M. (2012). *Introduction to the theory of computation*. Cengage learning.

References: finite-state automata and transducers I

Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., & Mohri, M. (2007). OpenFst: A general and efficient weighted finite-state transducer library [`http://www.openfst.org`]. In *Proceedings of the ninth international conference on implementation and application of automata, (ciaa 2007)*, Springer. `http://www.openfst.org`.

Beesley, K. R., & Karttunen, L. (2003). *Finite-state morphology: Xerox tools and techniques*. CSLI, Stanford.

Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computational Linguistics*, *23*(2), 269–311.

Mohri, M., Pereira, F., & Riley, M. (2002). Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, *16*(1), 69–88.

Roche, E., & Schabes, Y. (1997). *Finite-state language processing*. MIT Press.

References: phonotactics I

Futrell, R., Albright, A., Graff, P., & O'Donnell, T. J. (2017). A generative model of phonotactics. *Transactions of the Association for Computational Linguistics*, *5*, 73–86.

Hayes, B. (2011). *Introductory phonology* (Vol. 32). John Wiley & Sons.

Hayes, B., & Wilson, C. (2007). A maximum entropy model of phonotactics and phonotactic learning. *Linguistic Inquiry*, *39*, 379–440.

Heinz, J. (2010). Learning long-distance phonotactics. *Linguistic Inquiry*, *41*(4), 623–661.

McQueen, J. M. (1998). Segmentation of continuous speech using phonotactics. *Journal of Memory and Language*, *39*(1), 21–46.

References: weak and strong generative capacity I

Bresnan, J., Kaplan, R. M., Peters, S., & Zaenen, A. (1982). Cross-serial dependencies in dutch. In *The formal complexity of natural language* (pp. 286–319). Springer.

Chomsky, N., & Miller, G. A. (1963). Introduction to the formal analysis of natural languages. In R. D. Luce, R. R. Bush, & E. Galanter (Eds.), *Handbook of mathematical psychology* (pp. 269–321). New York: John Wiley & Sons, Inc.

Kornai, A., & Pullum, G. K. (1990). The x-bar theory of phrase structure. *Language*, 24–50.

Miller, P. (2000). *Strong generative capacity: The semantics of linguistic formalism*. Cambridge.

Vijay-Shanker, K., & Weir, D. J. (1994). The equivalence of four extensions of context-free grammar. *Mathematical Systems Theory*, *27*(6), 511–546.

References: limits of finite-state models for natural language I

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, *2*(3), 113–124.

Karlsson, F. (2007). Constraints on multiple center-embedding of clauses. *Journal of Linguistics*, 365–392.

Kornai, A. (1985). Natural languages and the Chomsky hierarchy. In *Second conference of the European chapter of the association for computational linguistics*, Geneva, Switzerland, Association for Computational Linguistics.

Miller, G. A., & Chomsky, N. (1963). Finitary models of language users. In R. D. Luce, R. R. Bush, & E. Galanter (Eds.), *Handbook of mathematical psychology* (pp. 419–491). New York: John Wiley & Sons, Inc.